SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3 DISTRIBUTION / AVAILABILITY OF REPORT Approved for public release; distribution unlimited.
2b. DECLASSIFICATION / DOWNGRADING SCHEDULE	
4 PERFORMING ORGANIZATION REPORT NUMBER(S) NRL Memorandum Report 5504	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION Naval Research Laboratory 6b. OFFICE SYMBOL (*If applicable*) Code 4040	7a. NAME OF MONITORING ORGANIZATION
6c. ADDRESS (*City, State, and ZIP Code*) Washington, DC 20375-5000	7b. ADDRESS (*City, State, and ZIP Code*)
8a. NAME OF FUNDING / SPONSORING ORGANIZATION 8b. OFFICE SYMBOL (*If applicable*)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (*City, State, and ZIP Code*)	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. (See page ii) / PROJECT NO. / TASK NO. / WORK UNIT ACCESSION NO.
11. TITLE (*Include Security Classification*) Vectorization and Implementation of an Efficient Multigrid Algorithm for the Solution of Elliptic Partial Differential Equations	
12. PERSONAL AUTHOR(S) DeVore, C.R.	
13a. TYPE OF REPORT Interim 13b. TIME COVERED FROM _____ TO _____	14. DATE OF REPORT (*Year, Month, Day*) 1984 December 31 15. PAGE COUNT 39
16. SUPPLEMENTARY NOTATION This research was supported by the Office of Naval Research and the Naval Research Laboratory.	
17 COSATI CODES FIELD / GROUP / SUB-GROUP	18. SUBJECT TERMS (*Continue on reverse if necessary and identify by block number*) Elliptic equations; Multigrid algorithms; Numerical methods; Vector computers

19. ABSTRACT (*Continue on reverse if necessary and identify by block number*)

Multilevel adaptive techniques are powerful methods for solving many types of problems in computational physics. In his analysis of multilevel methods for application to elliptic boundary-value problems, Douglas found a particularly efficient algorithm for their solution. I describe in this report a vectorized, finite-difference implementation of his algorithm suitable for use in large-scale computations.

20 DISTRIBUTION / AVAILABILITY OF ABSTRACT ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED
22a. NAME OF RESPONSIBLE INDIVIDUAL C. R. DeVore	22b. TELEPHONE (*Include Area Code*) (202) 767-2891 22c. OFFICE SYMBOL Code 4040

DD FORM 1473, 84 MAR 83 APR edition may be used until exhausted.
All other editions are obsolete.

SECURITY CLASSIFICATION OF THIS PAGE

10. SOURCE OF FUNDING NUMBERS

PROGRAM ELEMENT NO.	PROJECT NO.	TASK NO.	WORK UNIT ACCESSION NO.
61153N	RR014-03-OF		DN380-225
61153N	RR011-09-43		DN280-068

NRL Memorandum Report 5504

Vectorization and Implementation of an Efficient Multigrid Algorithm for the Solution of Elliptic Partial Differential Equations

C. R. DEVORE

Laboratory for Computational Physics

December 31, 1984

This research was supported by the Office of Naval Research
and the Naval Research Laboratory.

NAVAL RESEARCH LABORATORY
Washington, D.C.

CONTENTS

VECTORIZATION AND IMPLEMENTATION OF AN EFFICIENT MULTIGRID ALGORITHM FOR THE SOLUTION OF ELLIPTIC PARTIAL DIFFERENTIAL EQUATIONS

INTRODUCTION

Elliptic partial differential equations arise in a wide variety of problems in computational physics and engineering. These applications include, for example, the calculation of incompressible flows in hydrodynamics, electromagnetic potentials in magnetohydrodynamics, and molecular states in quantum chemistry. Elliptic equations also arise from time-implicit formulations of diffusion problems, after discretization of the time variable. For applications involving time-dependent physical systems, the equations must be solved over the spatial domain at each timestep. Over the duration of a typical simulation, this amounts to solving thousands of equations in hundreds or thousands of unknowns. Executing this imposing task in a practical fashion calls for the development and use of accurate and highly efficient numerical techniques. The importance of elliptic equations in applied mathematics has led to extensive and varied efforts in this direction [cf. Schultz 1981].

During the last decade, much attention has been devoted to the development of a new class of methods, multilevel adaptive techniques [cf. Brandt 1977], for solving many types of numerical problems. The general principles of multilevel techniques apply equally to finite-difference and finite-element approaches to solving partial differential equations, as well as to problems not associated with such equations, and can be summarized as follows. As for all numerical methods, one begins by specifying a discretization of the original (continuous) problem, and then seeks the solution to this discretized problem in the appropriate finite-dimensional

Manuscript approved November 1, 1984.

(physical or function) space. The essence of the multigrid approach is to establish a sequence of smaller, perhaps nested, auxiliary spaces, and to use solutions obtained in these smaller spaces to approximate the desired solution in the largest space. The solution in any space can be improved by combining relaxation iterations in that space, which smooth the fine-scale errors, with solving correction problems using the smaller spaces, which reduce the coarse-scale errors. The smaller spaces have geometrically fewer unknowns, and so require far less computational effort to yield a solution than does the largest space. The final result is a substantial savings in the work required to solve the problem.

Multigrid algorithms for elliptic boundary-value problems have been proposed and analyzed by several authors [cf. Douglas 1984, and references cited therein]. Douglas [1982] found a particularly efficient multigrid algorithm for solving elliptic problems, one which in numerical tests yielded solutions with accuracy comparable to that obtained by several alternative algorithms, for a smaller amount of effort. His algorithm has been implemented in finite-difference form in a manner suitable for use in large-scale numerical simulations on a high-speed vector computer. In this report I summarize the principles guiding its implementation, discuss the modifications to the algorithm required by these principles, and present the results of numerical tests of the method.

THE MULTIGRID ALGORITHM

Suppose we are given an elliptic equation to solve on the unit square $[0,1]\times[0,1]$ in the (x,y) plane. Lay a uniform $N\times N$ mesh on the unit square, and denote the mesh spacing by $h = 1/N$. After finite-differencing the differential equation and incorporating boundary conditions, the problem is reduced to solving the matrix equation

$$A^h u^h = f^h, \tag{1}$$

where A^h is the matrix of coefficients, u^h is the sought solution, and f^h is the inhomogeneous term, defined on the discrete domain D^h. We might attempt to solve Eq. (1) by a direct method such as sparse Gaussian elimination, or by an iterative method such as Gauss-Seidel relaxation. For the direct approach, the operation count is $O(N^3)$ to obtain a solution for the N^2 unknowns in Eq. (1), and thus rapidly becomes very large for large problems. For the iterative approach, the operation count is just proportional to the number of unknowns; owing to the slow relaxation of the coarse-scale errors in the solution, however, many iterations must be done in order to obtain an accurate result, and the operation count is again very large, because the coefficient of N^2 is large. Multigrid techniques take advantage of the rapid reduction in effort expended by direct methods as the number of unknowns is reduced, and of the rapid relaxation of fine-scale errors in the solution by iterative methods, by utilizing a combination of the two.

A simple multigrid algorithm for solving this elliptic problem is the following. Choose N to be even, and lay two uniform grids on the unit square: one N×N with mesh spacing h, the other $\frac{N}{2}\times\frac{N}{2}$ with mesh spacing 2h. On the fine grid D^h we have Eq. (1), while on the coarse grid D^{2h} we have

$$A^{2h}\, u^{2h} = f^{2h}. \tag{2}$$

We begin by solving Eq. (2) using a direct method. Since the number of unknowns is smaller by a factor of 4, this task requires a factor of $2^3 = 8$ less work than does the same task on D^h. We then interpolate the solution U^h to Eq. (2) onto D^h,

$$U^h = I^h_{2h}\, U^{2h}, \tag{3}$$

where I is an interpolation operator. The result U^h possesses fine-scale errors due to the lack of fine-scale information in A^{2h} and f^{2h}, and thus U^{2h}, as well as any introduced by the interpolation process. We therefore do a small number of relaxation iterations on U^h to smooth these fine-scale errors, and obtain

$$\tilde{U}^h = R\, U^h, \tag{4}$$

where R is a relaxation operator. $\tilde{U}^h$ is our first approximation $u^{h(1)}$ to the solution to Eq. (1).

We can improve the approximation to u^h by solving a correction problem. Define the residual r^h and the correction v^h on D^h by the expressions

$$r^h = f^h - A^h \tilde{U}^h \tag{5}$$

and

$$v^h = u^h - \tilde{U}^h, \tag{6}$$

respectively, so that v^h satisfies

$$A^h v^h = r^h. \tag{7}$$

Notice that Eq. (7) has the same form as Eq. (1). We now use a projection operator P to project r^h onto D^{2h},

$$r^{2h} = P_h^{2h} r^h, \tag{8}$$

and solve directly the coarse-grid equivalent of Eq. (7),

$$A^{2h} v^{2h} = r^{2h}, \tag{9}$$

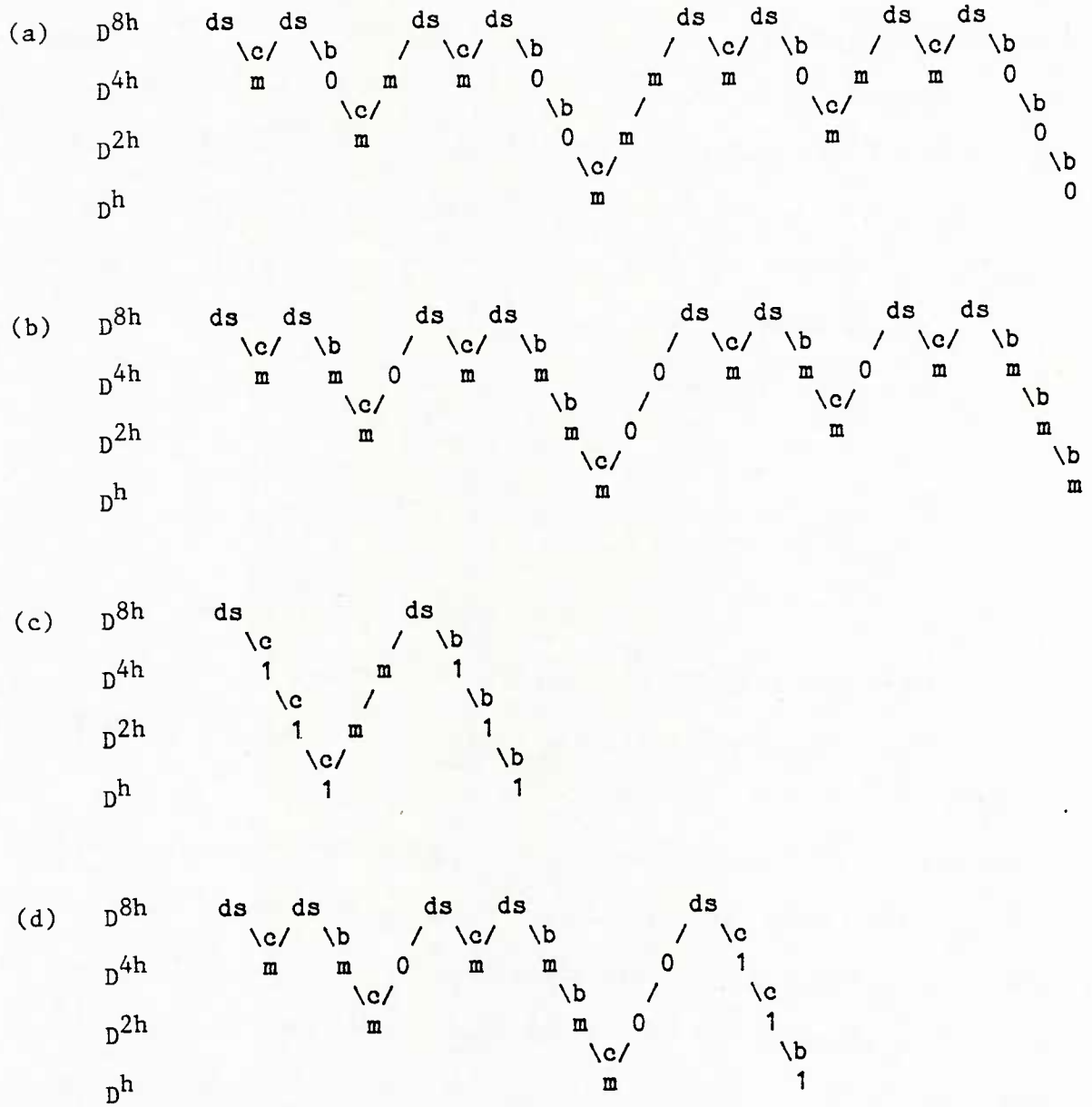
to obtain V^{2h}. We apply the interpolation operator I to the correction V^{2h}, and add the result V^h to $\tilde{U}^h$. Smoothing the fine-scale errors by applying the relaxation operator R, we obtain

$$u^{h(2)} = R(\tilde{U}^h + V^h), \tag{10}$$

our second approximation to the desired solution on the fine grid.

This simple algorithm is a 2-level, 2-cycle algorithm for solving elliptic boundary value problems. It can readily be extended to make use of additional coarse grids with mesh spacings 4h, 8h, etc. (more levels can be utilized) or to solve additional correction problems (more cycles can be done), or both. Further options to be exercised in the implementation of a multigrid algorithm include the selection of a direct solver and of a relaxation technique, and the specification of interpolation and projection schemes. The best choices for many of these options will in general depend upon the characteristics of the problem to be solved (the properties of the coefficients in the equation, the number of spatial dimensions, the number of unknowns, the accuracy required in the solution) and the characteristics of the computing machinery to be used (capabilities for vector or parallel processing, relative costs of execution and core storage time).

Several multilevel algorithms for elliptic boundary value problems have been suggested and analyzed by various authors, for both finite-element and finite-difference discretizations. I show schematically in Fig. 1 a 4-level, 2-cycle example of each of four algorithms. The first is due to Federenko [1961], the second is due to Brandt [1977], the third is an iterative extension by Brandt [1978] of a 2-level algorithm also due to Federenko [1961], and the fourth is a hybrid of the latter two schemes investigated by Douglas [1982]. The general N-cycle hybrid algorithm consists of N-1 cycles of the second algorithm followed by a half-cycle of the third. The first two algorithms are different in that in the first case smoothing iterations are done after projection, while in the second they are



ds = direct solve
m = m smoothing iterations
1 = one smoothing iteration
0 = no smoothing iterations
b = bilinear interpolation
c = cubic interpolation

FIGURE 1. Four multigrid algorithms for elliptic boundary value problems, illustrated by 4-level, 2-cycle examples.

done after interpolation; they are alike in that they are both recursive, i.e., they cycle at all levels save the lowest one. The third algorithm allows smoothing after both projection and interpolation, and is not recursive, cycling occurring only at the highest level. Douglas [1982, 1984] has shown that the recursive algorithms are optimal order algorithms, i.e., the operation count for a solve to truncation error is proportional to the number of unknowns, if the condition

$$2 \leq C < 2^{\delta} \tag{11}$$

is satisfied, where C is the number of cycles and δ is the number of space dimensions in the problem. For the case $C = 2^{\delta}$ (e.g., a 1-dimensional problem solved by a 2-cycle algorithm), the operation count for N unknowns is O(NlogN).

Douglas [1982] carried out extensive numerical experiments on these algorithms, using sparse Gaussian elimination for the direct solves and Gauss-Seidel relaxation for the smoothing iterations. He considered various schemes for the interpolation of the solutions and corrections and for the projection of the residuals. His results indicate that a combination of high-order (cubic) interpolation of the solutions and low-order (bilinear) interpolation of the corrections is optimal; high-order interpolation of the corrections often degrades the accuracy of the resulting 'corrected' solution, and additional smoothing iterations are necessary to compensate. He found it advantageous to smooth the residuals on projection by using a weighted average of the neighboring values on the finer grid. Both of these

findings reflect the purpose of the correction problem, which is to improve the accuracy in the coarse-scale features of the solution, leaving the fine-scale features to the relaxation scheme. Douglas further found that the hybrid algorithm yielded solutions with accuracy comparable to that obtained by Federenko's [1961] and Brandt's [1977] algorithms, for a smaller amount of work. By comparison, sparse Gaussian elimination on the finest grid yielded the same accuracy at a substantial increase in computational resources: a factor of 10 in operations and a factor of 4 in memory, for a problem in 1000 unknowns. Gauss-Seidel relaxation to convergence on the finest grid required an increase of a factor of 100 in operations over the multigrid algorithms for that problem.

VECTORIZATION OF THE ALGORITHM

Efficient computational techniques make optimal use of computer resources by minimizing some machine-dependent combination of execution and core storage time. Given that a consideration of the tradeoffs between execution and core storage for a particular technique establishes a minimum memory requirement for its efficient use, the task which remains is to minimize the execution time subject to this memory constraint. For a scalar (sequential) processor, which can execute one instruction on one scalar or pair of scalars at a time, this task is essentially identical to that of minimizing the total number of arithmetic operations performed. For a vector processor, which can execute a single instruction on all of the elements of a vector or pair of vectors serially, the direct correspondence between execution time and number of operations fails, owing to the marked difference in speed of execution (an order of magnitude or greater) of vector code vis-a-vis scalar code. This circumstance places a premium on the use of vectorizable instructions, even at the expense of a substantial increase in the total number of operations performed. Similar considerations apply to an n-component parallel processor, which can execute n different instructions on n independent vectors or pairs of vectors simultaneously; a larger operation count can be borne by a more highly parallel algorithm, at a net reduction in execution time.

Vectorization of several of the component operations of the multigrid algorithms, specifically the generation of the matrices, the interpolation of the solutions and corrections, and the calculation and projection of the

residuals, is straightforward. The direct solver is employed only on very coarse grids, to solve problems with few unknowns. Its operation count is therefore usually a negligible fraction of the total, and furthermore the vectors on which it operates are generally too short for a significant gain in speed by the processor, so that vectorization is not an overriding consideration in its selection. The largest share of the arithmetic operations executed by a multigrid algorithm is performed by the relaxation method, primarily on the finer grids possessing many unknowns. The advantages accruing to extensive vector processing thus figure prominently in the selection of a smoothing technique. A discussion of these considerations for parallel processing applications has been given by Brandt [1981].

A classical relaxation method eminently suited for use on a vector processor is Jacobi relaxation. Given the set of simultaneous equations

$$\sum_{\ell} M_{k\ell} X_{\ell} = Y_k, \tag{12}$$

where $M_{kk} \neq 0$, and an estimate X^n, the iteration reads

$$X_k^{n+1} = [Y_k - \sum_{\ell \neq k} M_{k\ell} X_{\ell}^n] \; / \; M_{kk}. \tag{13}$$

A more rapidly convergent method is Gauss-Seidel relaxation, in which the updated values of $X_{\ell<k}$ are employed on the right in Eq. (13), i.e.,

$$X_k^{n+1} = [Y_k - \sum_{\ell<k} M_{k\ell} X_{\ell}^{n+1} - \sum_{\ell>k} M_{k\ell} X_{\ell}^n] \; / \; M_{kk}. \tag{14}$$

The latter iteration, however, does not vectorize because the operands in the calculation of X_k depend on the results of the same instruction carried out for $X_{\ell<k}$.

A method which realizes an increase in the convergence rate by using updated values of X, but which vectorizes completely, is red-black or checkerboard relaxation. In this scheme, Eq. (13) is first applied to all 'red' (even k) points; then for all 'black' (odd k) points, calculate

$$X_k^{n+1} = [Y_k - \sum_{\text{even } \ell} M_{k\ell} X_\ell^{n+1} - \sum_{\text{odd } \ell} M_{k\ell} X_\ell^{n}] \; / \; M_{kk}. \qquad (15)$$

A potential shortcoming of red-black relaxation is that the retrieval and storage of the operands and the results is effectively reduced by about a factor of two due to the on-off pattern of the iteration. If the machine has a narrow memory bandwidth, the execution speed may be reduced sufficiently below that of Jacobi relaxation to negate most of the gain in the rate of convergence. This inefficiency can be circumvented by storing all 'red' points contiguously in memory, and likewise with all 'black' points, but at a substantial increase in complexity in other components of the solver. The simplicity and the ease of implementation of the Jacobi relaxation scheme may in many cases make it the superior method.

IMPLEMENTATION OF THE ALGORITHM

In numerical hydrodynamics calculations using finite differences, it is advantageous after subdividing the domain of the calculation to define the fluid variables at points interior to the subdomains (i.e., at the cell centers), as shown in Fig. 2(a) for a 2-dimensional problem. Fluxes are defined on the boundaries of the subdomains (i.e., at the cell interfaces) which describe the transport of mass, momentum, and energy from each cell to its neighbors. With the exception of the fluxes through the boundary of the system, no net change in the total mass, momentum, and energy of the system is effected, since a flux loss by one cell is balanced by a flux gain by its neighbor. The conservation of mass, momentum, and energy required by the laws of hydrodynamics is therefore achieved by the numerical solutions.

When applying multigrid techniques to solve elliptic boundary-value problems, on the other hand, it is advantageous to define the dependent variable on the cell interfaces, as shown in Fig. 2(b). The coarse-grid points are then coincident with points on the finest grid, and this coincidence simplifies the intergrid transfers (interpolations and projections) required by the multigrid algorithm. Douglas [1982] used such nested grids to obtain his numerical results.

Modifications to the intergrid transfers in Douglas' multigrid algorithm have been made to accomodate the change from cell-interface to cell-center locations for defining the dependent variable. Fig. 3 shows the spatial relationships between nine coarse-grid points and sixteen fine-grid points, and some of their associated cell interfaces. None of the coarse-

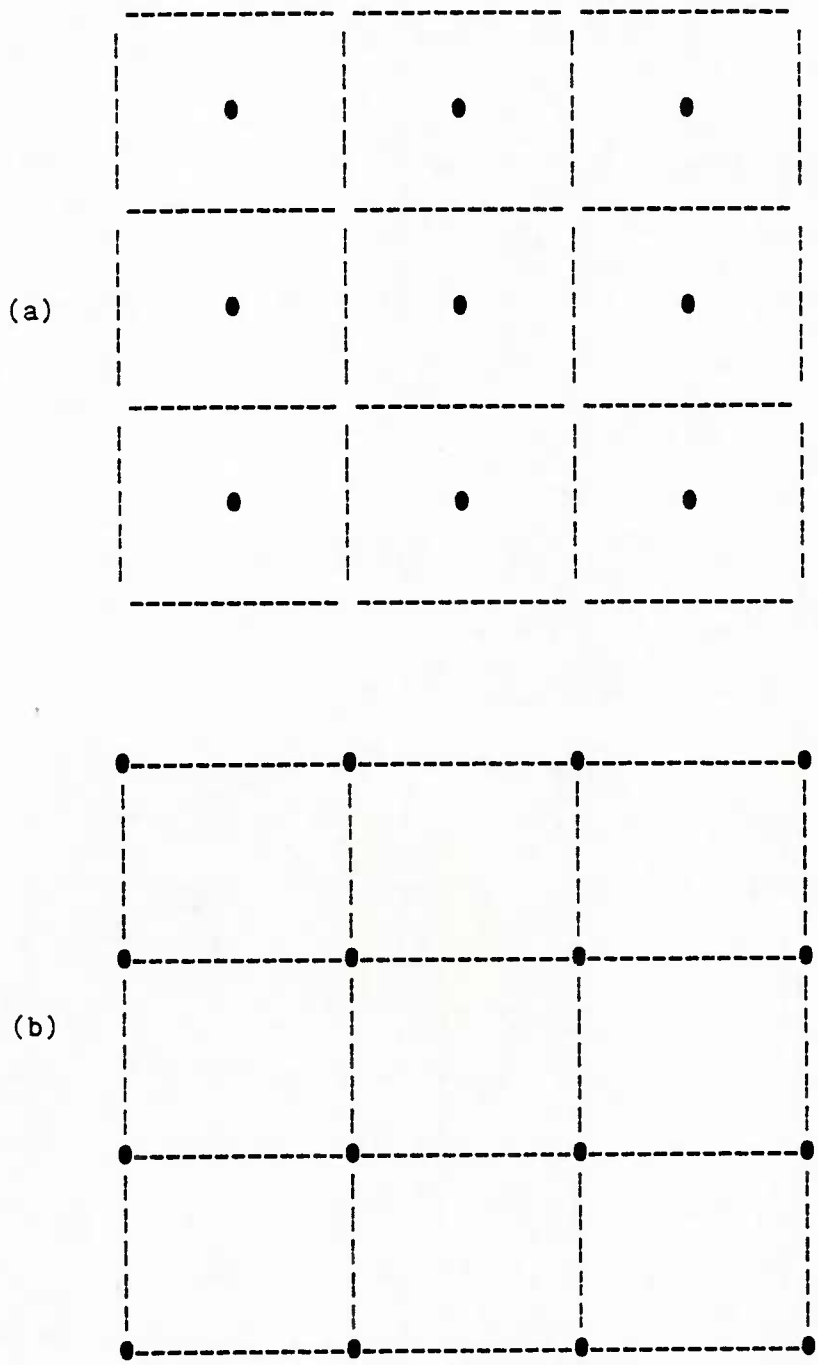


FIGURE 2. Subdivision of a 2-dimensional rectangular domain into $3^2 = 9$ subdomains. For hydrodynamic calculations, dependent variables are defined most conveniently at points interior to the subdomains, at cell centers (a). For multigrid solutions to elliptic problems, dependent variables are defined most conveniently at cell interfaces (b).

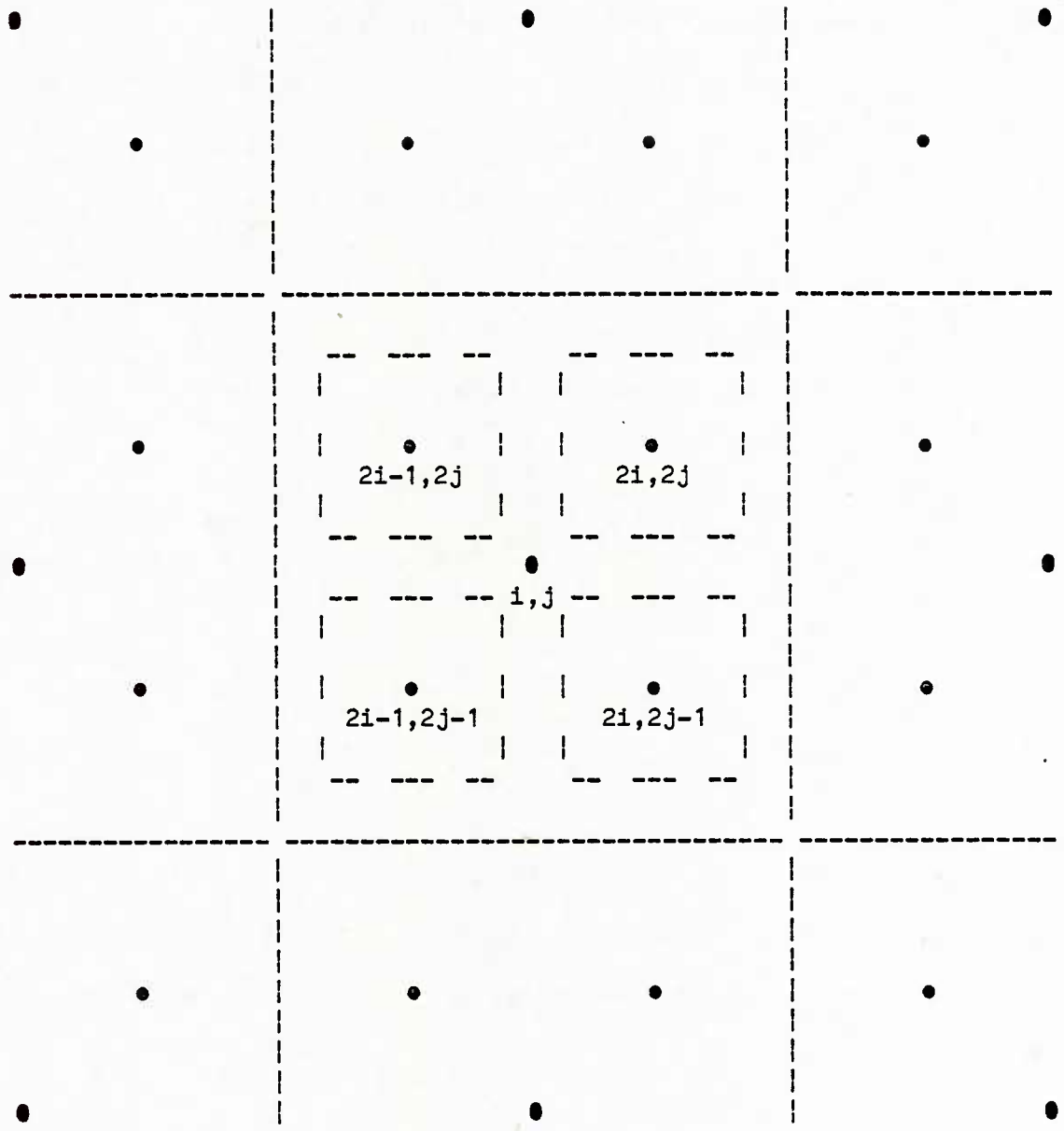


FIGURE 3. Spatial relationships between coarse-grid points (●) and fine-grid points (•), and between their associated cell interfaces. The indices of the one coarse-grid point and the four fine-grid points in the center of the figure are shown.

and fine-grid points are coincident. The fine-grid mesh spacings in the x- and y-directions are denoted by Δx and Δy, respectively.

The interpolation formulae are easily derived. Define for a given pair x_0, y_0

$$F_{\alpha\beta} \equiv F(x_0 + \alpha\Delta x, y_0 + \beta\Delta y), \tag{16}$$

for any function $F(x,y)$. A Taylor series expansion yields

$$F_{1/2\ 1/2} = \frac{15}{16} F_{00} + \frac{1}{16} F_{22} + \frac{3}{32} (F_{20} + F_{02} - F_{-20} - F_{0-2}) \tag{17}$$

to $O(\Delta x^3, \Delta y^3)$, and

$$F_{1/2\ 1/2} = \frac{1}{2} F_{00} + \frac{1}{4} (F_{20} + F_{02}) \tag{18}$$

to $O(\Delta x^2, \Delta y^2)$. The quadratic interpolation of Eq. (17) is applied to the solutions U, while the bilinear interpolation of Eq. (18) is applied to the corrections V. Thus, for example, referring to Fig. 3, at the fine-grid point (2i,2j) we have

$$\begin{aligned} U^h_{2i,2j} &= \frac{15}{16} U^{2h}_{i,j} + \frac{1}{16} U^{2h}_{i+1,j+1} \\ &+ \frac{3}{32} (U^{2h}_{i+1,j} + U^{2h}_{i,j+1} - U^{2h}_{i-1,j} - U^{2h}_{i,j-1}) \end{aligned} \tag{19}$$

and

$$V^h_{2i,2j} = \frac{1}{2} V^{2h}_{i,j} + \frac{1}{4} (V^{2h}_{i+1,j} + V^{2h}_{i,j+1}). \tag{20}$$

At the boundary of the domain, where the necessary coarse-grid points do not exist, quadratic or bilinear extrapolation is used, as appropriate. The cubic interpolation of the solutions employed by Douglas is rather unwieldy for use here, since the interpolation must be performed at every fine-grid point, in two dimensions. In his implementation, interpolation must be done only at the new (non-coincident) fine-grid points, and then primarily only in one dimension, x or y, since most of the new fine-grid points are collinear with the coarse-grid points.

The residual projection is accomplished by taking an equally-weighted average of the sixteen nearest fine-grid neighbors of each coarse-grid point. Again referring to Fig. 3, the residual projection at (i,j) is therefore given by

$$r^{2h}_{i,j} = \frac{1}{16} \sum_{k=2i-2}^{2i+1} \sum_{\ell=2j-2}^{2j+1} r^{h}_{k\ell}. \qquad (21)$$

At the boundary of the domain, those fine-grid residuals in the sum which are undefined (belonging to fine-grid points beyond the boundary) are just omitted from the average. Projection schemes which assign a larger weighting factor to the four inner neighbors than to the twelve outer ones did not perform as well as the equal-weighting scheme.

A further issue in the intergrid transfers not considered by Douglas is the projection of the coefficients and source terms in the difference equations. Consider the following 2-dimensional elliptic boundary-value problem, defined on the rectangular domain $D = [x_{min},x_{max}]\times[y_{min},y_{max}]$:

$$\frac{\partial}{\partial x}\left(P\frac{\partial u}{\partial x}\right) + \frac{\partial}{\partial y}\left(Q\frac{\partial u}{\partial y}\right) + \frac{\partial}{\partial x}(Ru) + \frac{\partial}{\partial y}(Su) + Tu = f \qquad (22)$$

in the interior, and

$$au + b\frac{\partial u}{\partial n} = c \qquad (23)$$

on the boundary (n is the coordinate normal to the boundary). Subdividing the domain into $N_1 \times N_2$ cells with mesh spacings Δx and Δy and finite-differencing Eqs. (22) and (23), we obtain

$$\frac{1}{\Delta x^2}\left[P_{i+1/2,j}\,(u_{i+1,j} - u_{i,j}) - P_{i-1/2,j}\,(u_{i,j} - u_{i-1,j})\right] +$$

$$\frac{1}{\Delta y^2}\left[Q_{i,j+1/2}\,(u_{i,j+1} - u_{i,j}) - Q_{i,j-1/2}\,(u_{i,j} - u_{i,j-1})\right] +$$

$$\frac{1}{2\Delta x}\left[R_{i+1/2,j}\,(u_{i+1,j} + u_{i,j}) - R_{i-1/2,j}\,(u_{i,j} + u_{i-1,j})\right] + \qquad (24)$$

$$\frac{1}{2\Delta y}\left[S_{i,j+1/2}\,(u_{i,j+1} + u_{i,j}) - S_{i,j-1/2}\,(u_{i,j} + u_{i,j-1})\right] +$$

$$T_{i,j}\,u_{i,j} = f_{i,j},$$

together with

$$\frac{a_{1/2,j}}{2}\,(u_{1,j} + u_{0,j}) + \frac{b_{1/2,j}}{\Delta x}\,(u_{1,j} - u_{0,j}) = c_{1/2,j} \qquad (25a)$$

and

$$\frac{a_{N_1+1/2,j}}{2}\,(u_{N_1+1,j} + u_{N_1,j}) + \frac{b_{N_1+1/2,j}}{\Delta x}\,(u_{N_1+1,j} - u_{N_1,j}) \tag{25b}$$

$$= c_{N_1+1/2,j}$$

at $x = x_{min}$ and x_{max}, respectively, and with corresponding expressions to Eqs. (25) at $y = y_{min}$ and y_{max}.

The quantities with half-integer indices in Eqs. (24) and (25) are defined at the cell interfaces, and those with integer indices at the cell centers. Each of the coarse-grid interfaces is coincident with two fine-grid interfaces (cf. Fig. 3). The projections of the cell-interface coefficients to the coarse grid are therefore carried out by averaging the two values on the fine grid, e.g.,

$$P^{2h}_{i+1/2,j} = \frac{1}{2}\,(P^{h}_{2i+1/2,2j} + P^{h}_{2i+1/2,2j-1}) \tag{26a}$$

and

$$P^{2h}_{i-1/2,j} = \frac{1}{2}\,(P^{h}_{2i-1/2,2j} + P^{h}_{2i-3/2,2j-1}) \tag{26b}$$

The projections of the cell-center coefficient T and source term f are done by averaging the values at the four surrounding fine-grid cell centers, e.g.,

$$T^{2h}_{i,j} = \frac{1}{4}\,(T^{h}_{2i-1,2j-1} + T^{h}_{2i-1,2j} + T^{h}_{2i,2j-1} + T^{h}_{2i,2j}). \tag{27}$$

A more distributed projection of the source f (e.g., the sixteen-neighbor average applied to the residuals) causes a loss of fine-scale information and degrades the accuracy of the solution obtained.

The multigrid algorithm as described thus far is clearly applicable to problems in which the domain is uniformly gridded. In many applications, however, nonuniform gridding in one or both dimensions is necessary in order to guarantee sufficient spatial resolution in some fraction of the domain without requiring a prohibitively large number of unknowns in the problem. A simple algebraic transformation is available which converts the given finite-difference equation on the nonuniform grid into a related equation on a uniform grid, and the latter can be solved by multigrid techniques. For problems in which the grid nonuniformities are mild, the resulting solution to the uniform-grid equation may be a satisfactory approximation to the solution to the original nonuniform-grid equation.

The transformation to the uniform-grid problem is found from the equivalent of Eqs. (24) and (25) on the nonuniformly gridded domain D. Denote the x-direction cell-interface and cell-center locations by $x_{i+1/2}$ ($i = 0,\ldots,N_1$) and x_i ($i = 0,\ldots,N_1+1$), respectively, where $x_{1/2} = x_{min}$ and $x_{N_1+1/2} = x_{max}$; the cell centers x_0 and x_{N_1+1} thus lie beyond the boundary of the domain. Define the local cell-interface and cell-center separations and the average over the grid, respectively, by

$$\Delta x_i \equiv x_{i+1/2} - x_{i-1/2}, \tag{28a}$$

$$\Delta x_{i+1/2} \equiv x_{i+1} - x_i, \tag{28b}$$

and

$$\overline{\Delta x} \equiv (x_{max} - x_{min}) / N_1. \tag{28c}$$

Similar expressions hold for the y-direction grid quantities. With these definitions, Eq. (24) generalizes to

$$\begin{aligned}
&\frac{1}{\Delta x_i}\left[\frac{P_{i+1/2,j}}{\Delta x_{i+1/2}}(u_{i+1,j} - u_{i,j}) - \frac{P_{i-1/2,j}}{\Delta x_{i-1/2}}(u_{i,j} - u_{i-1,j})\right] + \\
&\frac{1}{\Delta y_j}\left[\frac{Q_{i,j+1/2}}{\Delta y_{j+1/2}}(u_{i,j+1} - u_{i,j}) - \frac{Q_{i,j-1/2}}{\Delta y_{j-1/2}}(u_{i,j} - u_{i,j-1})\right] + \\
&\frac{1}{\Delta x_i}\left[\frac{R_{i+1/2,j}}{2}(u_{i+1,j} + u_{i,j}) - \frac{R_{i-1/2,j}}{2}(u_{i,j} + u_{i-1,j})\right] + \\
&\frac{1}{\Delta y_j}\left[\frac{S_{i,j+1/2}}{2}(u_{i,j+1} + u_{i,j}) - \frac{S_{i,j-1/2}}{2}(u_{i,j} + u_{i,j-1})\right] + \\
&\qquad T_{i,j}\, u_{i,j} = f_{i,j},
\end{aligned} \tag{29}$$

and Eqs. (25) to

$$\frac{a_{1/2,j}}{2}(u_{1,j} + u_{0,j}) + \frac{b_{1/2,j}}{\Delta x_{1/2}}(u_{1,j} - u_{0,j}) = c_{1/2,j} \tag{30a}$$

and

$$\frac{a_{N_1+1/2,j}}{2}\left(u_{N_1+1,j} + u_{N_1,j}\right) + \frac{b_{N_1+1/2,j}}{\Delta x_{N_1+1/2}}\left(u_{N_1+1,j} - u_{N_1,j}\right) \tag{30b}$$

$$= c_{N_1+1/2,j}.$$

Now multiply Eq. (29) by the factors $\frac{\Delta x_i}{\overline{\Delta x}}$ and $\frac{\Delta y_j}{\overline{\Delta y}}$, and define the following:

$$P^*_{i+1/2,j} \equiv \frac{\overline{\Delta x}}{\Delta x_{i+1/2}}\frac{\Delta y_j}{\overline{\Delta y}} P_{i+1/2,j} \qquad Q^*_{i,j+1/2} \equiv \frac{\Delta x_i}{\overline{\Delta x}}\frac{\overline{\Delta y}}{\Delta y_{j+1/2}} Q_{i,j+1/2}$$

$$R^*_{i+1/2,j} \equiv \frac{\Delta y_j}{\overline{\Delta y}} R_{i+1/2,j} \qquad S^*_{i,j+1/2} \equiv \frac{\Delta x_i}{\overline{\Delta x}} S_{i,j+1/2}$$

$$T^*_{i,j} \equiv \frac{\Delta x_i}{\overline{\Delta x}}\frac{\Delta y_j}{\overline{\Delta y}} T_{i,j} \qquad f^*_{i,j} \equiv \frac{\Delta x_i}{\overline{\Delta x}}\frac{\Delta y_j}{\overline{\Delta y}} f_{i,j}$$

$$b^*_{1/2,j} \equiv \frac{\overline{\Delta x}}{\Delta x_{1/2}} b_{1/2,j} \qquad b^*_{N_1+1/2,j} \equiv \frac{\overline{\Delta x}}{\Delta x_{N_1+1/2}} b_{N_1+1/2,j}$$

$$a^* \equiv a \qquad c^* \equiv c$$

The result of these operations is to put Eqs. (29) and (30) into the form of Eqs. (24) and (25), with Δx and Δy replaced by $\overline{\Delta x}$ and $\overline{\Delta y}$ and with the coefficients replaced by their asterisked counterparts.

A final note on implementation concerns Poisson equations on a uniformly gridded domain. For this special case, P=Q=1 and R=S=T=0, the coefficients in the finite-difference equation, Eq. (24), are independent of the indices i and j. Because of this, the core storage required to solve the problem is vastly reduced and the operation count for the solve is also decreased significantly. These advantages have been exploited in a solver specifically designed to solve Poisson equations on a uniform grid. When solutions to Poisson equations on a nonuniform grid are required, the transformations above can be carried out and a general elliptic equation solver used.

NUMERICAL TESTS

The multigrid algorithm proposed by Douglas [1982] and illustrated in Fig. 1(d) has been vectorized and implemented as described in the preceding two sections, in a FORTRAN program for use on the Naval Research Laboratory's Texas Instruments Advanced Scientific Computer (ASC) [Texas Instruments Incorporated 1975]. The ASC is a 32-bit word, 2-pipe vector computer with one million words of high-speed central memory. Each of the two pipes, which consist of one memory buffer unit and one arithmetic unit per pipe, can produce eight words of output data per 80 ns central processor clock cycle, under optimum conditions. The memory control unit can access one eight-word octet of input data per 160 ns central memory clock cycle. At peak efficiency, the ASC is capable of retrieving 50 million operands per second and, thus, of generating up to 50 million results per second. Since the two pipes are in principle able to execute as many as 200 million operations per second, for highly vectorized code the execution speed is limited by the memory bandwidth. A premium is therefore placed on the efficient retrieval and storage of data in optimizing ASC software.

The limiting of the execution speed by the memory bandwidth influences the selection of the relaxation algorithm, as discussed earlier. The red-black relaxation makes use of one-half of the eight words in each octet retrieved from memory, and as a result executes at roughly half of the speed of the Jacobi relaxation, which makes use of the entire octet. All of the results presented here were obtained using the Jacobi relaxation.

The direct solver for these tests uses a Crout lower-upper triangular

decomposition of the matrix. Parts of the solver are vectorized, and it takes advantage of the banded structure of the matrices arising from partial differential equations. The solver was adapted from a routine written by Winsor [1976].

The first set of test results is for the boundary-value problem on the unit square,

$$\frac{\partial}{\partial x}(e^{xy}\frac{\partial u}{\partial x}) + \frac{\partial}{\partial y}(e^{xy}\frac{\partial u}{\partial y}) + \frac{\partial}{\partial x}(yu) + \frac{\partial}{\partial y}(xu) + (x^2 + y^2)u = f \quad (31)$$

in the interior, and

$$u = 0 \quad (32)$$

on the boundary. The solution is chosen to be

$$u(x,y) = \sin(2\pi x)\sin(\pi y), \quad (33)$$

and the source term f(x,y) is defined by substituting this solution into the left side of Eq. (31). Some results are summarized in Table 1. There, for three different grid resolutions (16×16, 32×32, and 64×64) and for L-level, C-cycle solves for several values of L and C, are tabulated the number of floating-point operations performed per fine-grid point, the solve time per point in μs, and the maximum absolute error between the numerical solution and the exact solution (Eq. (33)) evaluated at the grid points. Also included is the storage required by the solver for each resolution and each value of L, given as the number of fine-grid arrays used. The results of

direct solves on the finest grid (L=C=1) are tabulated for the two lower resolution cases (16×16 and 32×32), there being insufficient memory to carry out the direct solve for the highest resolution case (64×64). For the first C-1 multigrid cycles, four Jacobi relaxation sweeps were done after each interpolation (m=4 in Fig. 1(d)); the error decreases only very slowly with additional sweeps, but increases quite rapidly with fewer sweeps.

The core storage M and the operation count W for the multigrid solver asymptote as L increases in Table 1. At each level, five arrays of storage are needed for the five bands of coefficients in the matrices, and one each is needed for the source term and for the solution. An additional array at each level is used to store the reciprocals of the matrix diagonal elements. This allows a (slow) divide to be replaced by a (fast) multiply in each relaxation iteration. Two further fine-grid arrays are needed, one for the residual calculation and another as general-purpose work space. The total storage required therefore asymptotes to

$$M = (1 + \frac{1}{4} + \ldots)\cdot 8 + 2 = \frac{4}{3}\cdot 8 + 2 = 12.7.$$

The asymptotic operation count per cell for the matrix generation is 35, and for the solve itself is 120 for the 2-cycle solve and 405 for the 3-cycle solve. The total operation counts therefore asymptote to

$$W_{C=2} = 35 + 120 = 155$$

and

$$W_{C=3} = 35 + 405 = 440$$

TABLE 1. Elliptic test problem results obtained on (a) 16×16, (b) 32×32, and (c) 64×64 grids. For each L-level, C-cycle solve (L=C=1 is direct solve), the operation count per cell, the solve time in μs per cell, and the maximum absolute error are tabulated. For each L, the core storage required in number of fine-grid arrays is given. Insufficient memory was available to solve the 64×64 problem directly (*).

	Cycles	Levels 1	2	3	4	5
(a)	1	617 operations/cell 120 μs/cell 2.9×10^{-5} maximum error				
	2		184 60 4.7×10^{-3}	139 51 6.7×10^{-3}		
	3			254 95 1.5×10^{-3}		
	Storage	47.1	17.7	14.1		
(b)	1	2233 266 8.3×10^{-5}				
	2		396 70 1.5×10^{-3}	161 34 3.0×10^{-3}	147 31 3.5×10^{-3}	
	3			307 66 2.6×10^{-4}	305 77 3.2×10^{-4}	
	Storage	78.6	21.0	13.8	13.5	
(c)	1	8538 * *				
	2		1204 141 3.3×10^{-4}	241 36 6.9×10^{-4}	156 23 9.2×10^{-4}	151 22 1.0×10^{-3}
	3			494 74 5.6×10^{-5}	342 56 4.6×10^{-5}	345 66 6.6×10^{-5}
	Storage	142.3	28.6	14.2	13.0	13.1

The tabulated values for the latter are significantly smaller, owing to the slow convergence of the operation count to its asymptotic limit as L increases.

The advantages of the multigrid solve over the direct solve in core storage, operation count, and execution time are clearly evident for all three cases, particularly the two higher resolution problems. Although the 16×16 multigrid solves are substantially less accurate than the direct solve, the 3-cycle solves on the 32×32 grid are comparable in accuracy to the direct solve, and those on the 64×64 grid exceed the inferred accuracy of the direct solve. The accuracy of the multigrid solutions varies rather slowly with L, but improves markedly with an increase in C, at the price of an increase of roughly a factor of two in operation count and execution time for the examples shown. The effect of decreasing vector length on the speed of execution of the algorithm is apparent in the 3-cycle solves for the 32×32 and 64×64 problems. For L = 3 and 4 and L = 4 and 5, respectively, the operation count is essentially identical, but the execution time is greater by over 15%, for the larger values of L compared to the smaller values. For the 2-cycle solves, on the other hand, the execution time is slightly smaller for the larger values of L. These results suggest the simple rule of thumb that the coarsest grid used be approximately 8×8, for either 2- or 3-cycle solves, in order to roughly minimize the execution time. Then, for example, a diffusion problem in 1000 unknowns can be accurately solved for 1000 timesteps at just over one minute of CPU time.

The second set of test results, summarized in Table 2, is for the Poisson boundary-value problem

TABLE 2. Poisson test problem results obtained on (a) 16×16, (b) 32×32, and (c) 64×64 grids. For each L-level, C-cycle solve (L=C=1 is direct solve), the operation count per cell, the solve time in μs per cell, and the maximum absolute error are tabulated. For each L, the core storage required in number of fine-grid arrays is given. Insufficient memory was available to solve the 64×64 problem directly (*).

	Cycles	Levels 1	2	3	4	5
(a)	1	595 operations/cell 113 μs/cell 2.9×10^{-5} maximum error				
	2		147 53 4.7×10^{-3}	99 45 5.9×10^{-3}		
	3			208 95 1.2×10^{-3}		
	Storage	40.8	9.5	5.4		
(b)	1	2210 262 8.3×10^{-5}				
	2		357 65 1.5×10^{-3}	117 28 2.8×10^{-3}	102 27 3.2×10^{-3}	
	3			250 61 2.3×10^{-4}	246 77 2.8×10^{-4}	
	Storage	72.4	13.1	5.4	5.1	
(c)	1	8514 * *				
	2		1164 137 3.2×10^{-4}	194 31 6.5×10^{-4}	108 18 8.8×10^{-4}	103 18 1.0×10^{-3}
	3			431 69 5.8×10^{-5}	273 52 3.4×10^{-5}	274 65 4.3×10^{-5}
	Storage	136.2	20.9	6.2	4.9	4.9

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = f \qquad (34)$$

on the unit square, again with the boundary condition

$$u = 0 \qquad (35)$$

and the solution

$$u(x,y) = \sin(2\pi x)\sin(\pi y). \qquad (36)$$

As before, four Jacobi relaxation sweeps were performed after each interpolation in the first C-1 multigrid cycles.

The core storage required by the Poisson solver is smaller by six arrays at each level - the five bands of coefficients in the matrices and the reciprocals of the diagonal elements - so that it asymptotes to

$$M = \frac{4}{3} \cdot 2 + 2 = 4.7.$$

The operation count for the matrix generation is negligible, and the asymptotic operation count per cell for the solve is reduced to

$$W_{C=2} = 100$$

and

$$W_{C=3} = 330.$$

The savings come in the relaxation iterations and the residual calculations and arise from the symmetry of the matrices about the diagonal. This reduction in the operation count is not, however, matched by a comparable

reduction in the execution time. A breakdown of the timing by components shows that the relaxation iterations and residual calculations, although they require fewer operations, require about as much time to execute as for the elliptic solver. This is due to a combination of less efficient retrieval and storage of data and of more effort expended on short vectors by these components of the Poisson solver. The benefit of eliminating the storage of the bands of matrix coefficients is obtained at the cost of making these portions of the solver more cumbersome and slow. Clearly a net gain is realized, however, primarily in the amount of core required, though secondarily also in the total execution time.

CONCLUSIONS

The numerical tests summarized in the preceding section demonstrate conclusively the power of multigrid techniques in obtaining numerical solutions to elliptic boundary-value problems. These methods may make practical the solution of problems which heretofore would have been considered prohibitively expensive or even impossible to attempt, particularly when extended to a larger number of spatial dimensions. It may also prove possible to implement multigrid techniques with a sufficient degree of parallelism to make them highly effective for use on systems of parallel processors. Perhaps the major drawback of these methods is the large startup effort required. The complexity of multigrid algorithms, with their intergrid transfers and recursive cycling, makes the development and debugging of a working program a time-consuming task. The potential payoff is sufficiently great, however, to warrant the investment.

ACKNOWLEDGMENTS

It is a pleasure to acknowledge the invaluable contributions to this project of Craig C. Douglas, who provided a copy of his original computer program and guidance in its use, of David E. Fyfe, whose experience in numerical analysis and numerical linear algebra was frequently and freely shared, and of Jay P. Boris, who suggested this endeavor in the beginning and established the nonuniform grid generalization along the way. This research was supported by the Office of Naval Research (DN380-225 61153N RR014-03-0F) and by the Naval Research Laboratory (DN280-068 61153N RR011-09-43).

REFERENCES

Brandt, A.: Math. Comput. 31 (1977) 333.

Brandt, A.: Lecture notes, ICASE Workshop on Multigrid Methods, ICASE, NASA Langley Research Center, 1978.

Brandt, A.: Elliptic Problem Solvers (M. H. Schultz, ed.), Academic Press, New York, 1981, p. 39.

Douglas, C. C.: Yale University Technical Report No. 223, 1982.

Douglas, C. C.: SIAM J. Numer. Anal. 21 (1984) 236.

Federenko, R.P.: Z. Vycisl. Mat. i. Mat. Fiz. 1 (1961) 922.

Schultz, M. H. (ed.): Elliptic Problem Solvers, Academic Press, New York, 1981.

Texas Instruments Incorporated: Description of the ASC System, Austin, 1975.

Winsor, N.: Scientifi[illegible]rogram Library, Naval Research Laboratory Research Computation Division, Washington, 1976.